Github Link:https://github.com/Gowtham0504/Phase-2.git

PROJECT TITLE:***Cracking the Market Code: AI-Driven Stock Price Prediction Using Time Series Analysis***

PHASE-3

1.Problem Statement

Objective:

The goal of this project is to develop a robust AI-driven system capable of predicting short- and mid-term stock price movements using time series analysis techniques. By leveraging deep learning models such as LSTM (Long Short-Term Memory) and Transformer-based architectures, this project aims to identify patterns in historical stock data and generate accurate, data-driven forecasts that can assist traders and investors in making informed decisions.

Problem:

Stock price forecasting remains a highly challenging task due to the volatile, non-linear, and non-stationary nature of financial markets. Traditional statistical models struggle to capture complex temporal dependencies and react to real-time market events. The challenge lies in building a predictive model that can:

- ✓ Effectively learn from historical price and volume data,
- ✓ Incorporate external factors (e.g., news sentiment, macroeconomic indicators),
- ✓ Handle noise and avoid overfitting,

✓ Generalize well to unseen market conditions.

Scope:

- ✓ Cracking the Market Code: AI-Driven Stock Price Prediction Using Time Series Analysis
- ✓ Data Source: Historical price data from sources like Yahoo Finance or Alpha Vantage.
- ✓ Target: Predict closing prices or returns for a selected set of stocks.
- ✓ Models: Compare classical time series models (e.g., ARIMA) with AI models (e.g., LSTM,Transformer).
- ✓ Evaluation: Use metrics such as RMSE, MAE, and directional accuracy.

2. Abstract

Introduction:
- Overview of stock market volatility
- Importance of accurate stock price prediction

Traditional Forecasting Methods:
- Brief on statistical models like ARIMA
- Limitations in handling non-linear patterns

Emergence of AI in Finance:
- Role of AI in transforming financial analytics
- Advantages over traditional methods

Understanding Time Series Data:
- Components: trend, seasonality, noise
- Challenges in modeling financial time series

Machine Learning Techniques:
- Supervised vs. unsupervised learning
- Algorithms: Linear Regression, Random Forest, SVM

Deep Learning Approaches:
- Introduction to Neural Networks
- Focus on LSTM networks for sequential data

Data Preprocessing
- Importance of data cleaning
- Handling missing values and outliers

Feature Engineering
- Selecting relevant features
- Incorporating technical indicators

Model Training and Validation
- Splitting data into training and test sets
- Cross-validation techniques

Evaluation Metrics
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Percentage Error (MAPE)

Case Study
- Application of LSTM on historical stock data
- Comparison with traditional models

Challenges and Limitations
- Overfitting in complex models
- Data quality and availability issues

Future Directions
- Incorporating sentiment analysis
- Real-time prediction systems

Conclusion
- Recap of key findings
- Implications for investors and analysts

3. System Requirements

Minimum System Requirements (For Small to Medium-Scale Models)

- **CPU**: Intel Core i5 (10th Gen or later) / AMD Ryzen 5 (3rd Gen or later)
- **RAM**: 16 GB
- **GPU**: NVIDIA GTX 1660 (6GB VRAM) or RTX 2060
- **Storage**: 512 GB SSD
- **OS**: Windows 10 / Ubuntu 20.04 / macOS 12 or later
- **Software/Frameworks**:
    - Python 3.9+
    - Libraries: numpy, pandas, matplotlib, scikit-learn, statsmodels
    - Deep Learning: TensorFlow 2.x or PyTorch
    - Time Series: prophet, tsfresh, sktime
    - IDE: VS Code / JupyterLab
    - Optional: Docker, Anaconda

Recommended System Requirements (For Deep Learning & Faster Training)

- **CPU**: Intel Core i7 (12th Gen or later) / AMD Ryzen 7 (5th Gen or later)
- **RAM**: 32 GB (especially for processing long time series)
- **GPU**: NVIDIA RTX 3060 Ti (8GB VRAM) or better (e.g., RTX 4070/4080 for faster deep learning)
- **Storage**: 1 TB SSD (fast I/O for large datasets)
- **OS**: Ubuntu 22.04 LTS (preferred for ML pipelines) / Windows 11
- **Software/Frameworks**:
    - Python 3.11+
    - Libraries: pandas, numpy, scikit-learn, statsmodels, matplotlib, seaborn
    - Deep Learning: PyTorch (with CUDA 11.x) or TensorFlow 2.15+

- Time Series: prophet, sktime, darts
- Data Handling: dask, modin (if datasets are huge)
- GPU drivers & CUDA toolkit installed
- JupyterLab / VS Code / PyCharm
- Version Control: git

Optional but Useful for Production-Level System

- **Cloud GPU Option**: AWS (EC2 with NVIDIA A100), Google Cloud, Azure ML
- **Database**: PostgreSQL / MySQL / TimescaleDB (for time series data storage)
- **Big Data**: Apache Spark (if dealing with massive stock tick data)
- **Model Deployment**: Docker + FastAPI/Streamlit + Kubernetes (for live prediction service)
- **Monitoring**: MLflow / Weights & Biases

Extra Tools for Stock Market AI Project

- Financial APIs: Yahoo Finance API, Alpha Vantage, IEX Cloud
- Backtesting libraries: backtrader, bt, zipline
- Technical Indicators: ta-lib, finta

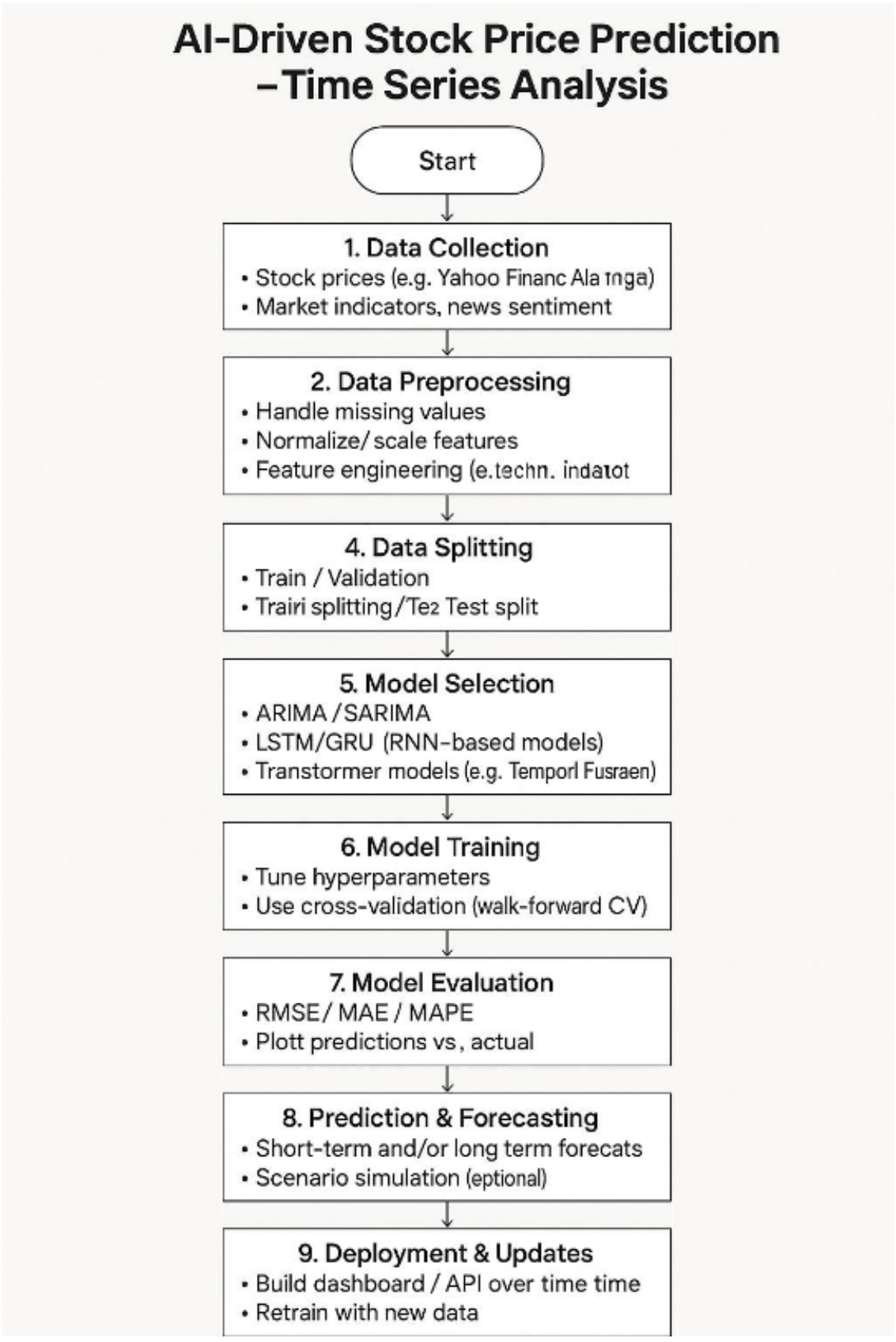
4. Objectives

- To collect and preprocess historical stock market data
- Acquire time series data (e.g., open, high, low, close, volume) from financial APIs and clean, normalize, and format it for model training.
- To explore and analyze stock price trends and patterns
- Perform exploratory data analysis (EDA) to understand price movements, detect

- seasonality, and identify market anomalies.
- To engineer relevant features from time series and external data
- Generate technical indicators (e.g., moving averages, RSI), lagged variables, and
- incorporate sentiment or macroeconomic indicators as additional inputs.
- To implement and compare multiple predictive models Develop and evaluate classical models (ARIMA, SARIMA) and AI models (LSTM, GRU,
- Transformer) for predicting future stock prices.
- To evaluate model performance using appropriate metrics
- Use performance measures like RMSE, MAE, and directional accuracy to assess model
- effectiveness and ensure reliable forecasts.
- To build a deployable prototype for real-time or batch predictions
- Create a functional pipeline or dashboard that visualizes forecasts and could be adapted for live market data.
- To assess the potential and limitations of AI in financial forecasting

- Analyze model behavior under different market conditions and highlight challenges such as overfitting, data noise, and response to unexpected events.

5. Flowchart of the Project Workflo


AI-Driven Stock Price Prediction –Time Series Analysis
Start
1. Data Collection
• Stock prices (e.g. Yahoo Financ Ala ıngа)
• Market indicators, news sentiment
2. Data Preprocessing
• Handle missing values
• Normalize/ scale features
• Feature engineering (e.techn. indatot
4. Data Splitting
• Train / Validation
• Trairi splitting/Te2 Test split
5. Model Selection
• ARIMA /SARIMA
• LSTM/GRU (RNN-based models)
• Transtormer models (e.g. Temporl Fusraen)
6. Model Training
• Tune hyperparameters
• Use cross-validation (walk-forward CV)
7. Model Evaluation
• RMSE/ MAE / MAPE
• Plott predictions vs, actual
8. Prediction & Forecasting
• Short-term and/or long term forecats
• Scenario simulation (eptional)
9. Deployment & Updates
• Build dashboard / API over time time
• Retrain with new data

6.Data Description: AI-Driven Stock Price Prediction (Time Series Analysis)

1. Stock Price Data

Source: Yahoo Finance, Alpha Vantage, Quandl, or similar.

Frequency: Daily (can also use intraday, weekly, or monthly depending on goal).

Fields:

Date: Timestamp of the observation.

Open: Price at market open.

High: Highest price during the day.

Low: Lowest price during the day.

Close: Final price when the market closes.

Adj Close: Adjusted closing price (accounts for splits/dividends).

Volume: Number of shares traded during the day.

2. Derived Features (Optional but recommended)

Technical Indicators (from libraries like TA-Lib or manually engineered):

SMA: Simple Moving Average.

EMA: Exponential Moving Average.

RSI: Relative Strength Index.

MACD: Moving Average Convergence Divergence.

Bollinger Bands: Price volatility bands.

Lagged Values:

Close_t-1, Close_t-2, etc., to help models capture temporal dependencies.

3. Sentiment or External Data (Optional)

News Headlines or Tweets (with sentiment score using NLP models).

Market Indices (e.g., S&P 500, Nasdaq).

Economic Indicators (e.g., interest rates, inflation data).

4. Target Variable

Usually:

Future Close Price (regression)

Price Movement Direction (classification: up/down)

7.Data Processing: AI-Driven Stock Price Prediction (Time Series Analysis)

1. Data Cleaning

• Remove duplicates: Ensure each date has only one row.

• Handle missing values:

• Fill with forward-fill (ffill), backward-fill (bfill), or interpolation.

• Drop rows if missing critical data like prices or volume.

2. Date Parsing & Indexing

• Convert Date column to datetime format.

• Set Date as the index for time series modeling.

3. Feature Engineering

• Lag features: Include past values as features (e.g., Close_t-1, Close_t-2, ...).

• Rolling statistics:

• Moving average (e.g., 7-day, 14-day, 30-day)

• Rolling standard deviation or volatility

• Technical indicators:

• SMA, EMA, RSI, MACD, Bollinger Bands

• Use ta-lib or pandas-ta for easy computation

4. Target Variable Creation

• Regression: Predict future Close price (e.g., Close_t+1, Close_t+5)

• Classification: Predict direction (1 for up, 0 for down)

5. Data Normalization/Scaling

• Use MinMaxScaler or StandardScaler (especially important for neural networks like

LSTM)

• Apply scaling to features only, not the date or target variable during model training.

6. Train-Test Split (Chronologically)

• Use 70–80% for training, remaining for testing or validation.

• Avoid random splitting to preserve temporal integrity.

📈 **1. Load the Data**

Make sure your data includes:

• Date/Time (Index)

• Open, High, Low, Close, Volume

• Possibly technical indicators (optional at this stage)

📊 **2. Basic Statistical Summary**

Understand the central tendencies and spread of the price and volume data. 📅

3. Time Series Plots

Visualize the trend and seasonality.

🔁 **4. Rolling Statistics (Moving Averages)**

Helps smooth out fluctuations and identify trend direction.

🌪️ 5. Volatility Analysis

Volatility can signal risk and opportunity.

🔗 6. Correlation Heatmap

Evaluate relationships between numerical columns.

🔍 7. Stationarity Check (ADF Test)

Stationary data is key for many models (ARIMA, SARIMA).

☑️ Next Steps After EDA:

• Feature Engineering (e.g., RSI, MACD)

• Train-test split

• Normalize/scale data

• Apply models: LSTM, Prophet, ARIMA, XGBoost

8.Feature Engineering for Stock Price Prediction

☑️ Steps for EDA in Stock Price Time Series Analysis

1. Load the Data

- Use stock price datasets (from Yahoo Finance, Alpha Vantage, etc.).
- Common columns: Date, Open, High, Low, Close, Adj Close, Volume.

```
import pandas as pd

df = pd.read_csv('stock_prices.csv', parse_dates=['Date'], index_col='Date')

print(df.head())
```

2. Check Missing Values & Data Types

- Stock datasets often have missing data on weekends/holidays.

```
print(df.info())

print(df.isnull().sum())
```

3. Visualize Stock Price Trends (Line Plots)

- Plot the **closing price** over time to see overall trends.

```
import matplotlib.pyplot as plt

df['Close'].plot(figsize=(12,6), title='Stock Closing Price Over Time')

plt.show()
```

4. Moving Averages (Smoothing)

- Use *Simple Moving Averages* (SMA) to smooth short-term fluctuations.

```
df['SMA50'] = df['Close'].rolling(window=50).mean()

df['SMA200'] =
df['Close'].rolling(window=200).mean()

df[['Close', 'SMA50', 'SMA200']].plot(figsize=(12,6))

plt.show()
```

5. Volatility Analysis

- Use daily returns or percentage changes to analyze stock volatility.

```
df['Daily Return'] = df['Close'].pct_change()

df['Daily Return'].hist(bins=50, figsize=(10,6))
```

```
plt.show()
```

6. Seasonality & Trend Decomposition

- Decompose time series into **trend**, **seasonality**, and **residual**.

```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(df['Close'], model='multiplicative', period=30)

result.plot()

plt.show()
```

7. Correlation Heatmap

- Correlation between *Open, High, Low, Close, Volume*.

```
import seaborn as sns
```

```
plt.figure(figsize=(8,6))

sns.heatmap(df.corr(), annot=True,
cmap='coolwarm')

plt.show()
```

8. Lag Plots (Autocorrelation)

- To check if past values influence future ones (good for time series models).

```
from pandas.plotting import lag_plot

lag_plot(df['Close'])

plt.show()
```

9. Stationarity Test (Augmented Dickey-Fuller Test)

- Important before applying ARIMA/LSTM models.

```
from statsmodels.tsa.stattools import adfuller
```

result = adfuller(df['Close'].dropna())

print(f'ADF Statistic: {result[0]}')

print(f'p-value: {result[1]}')

10. Volume vs Price Analysis

- Check if high trading volumes impact stock price changes.

df[['Volume', 'Close']].plot(subplots=True, figsize=(10,6))

plt.show()

9. Feature Engineering for Stock Price Prediction

1. Lag Features

These are past values used to predict the future.

2. Rolling Window Statistics

These show trends and volatility over time.

📊 3. Returns

Helps identify momentum or reversals.

4. Technical Indicators

- Popular indicators used in trading.
- Relative Strength Index (RSI)
- Moving Average Convergence Divergence (MACD)
- Bollinger Bands

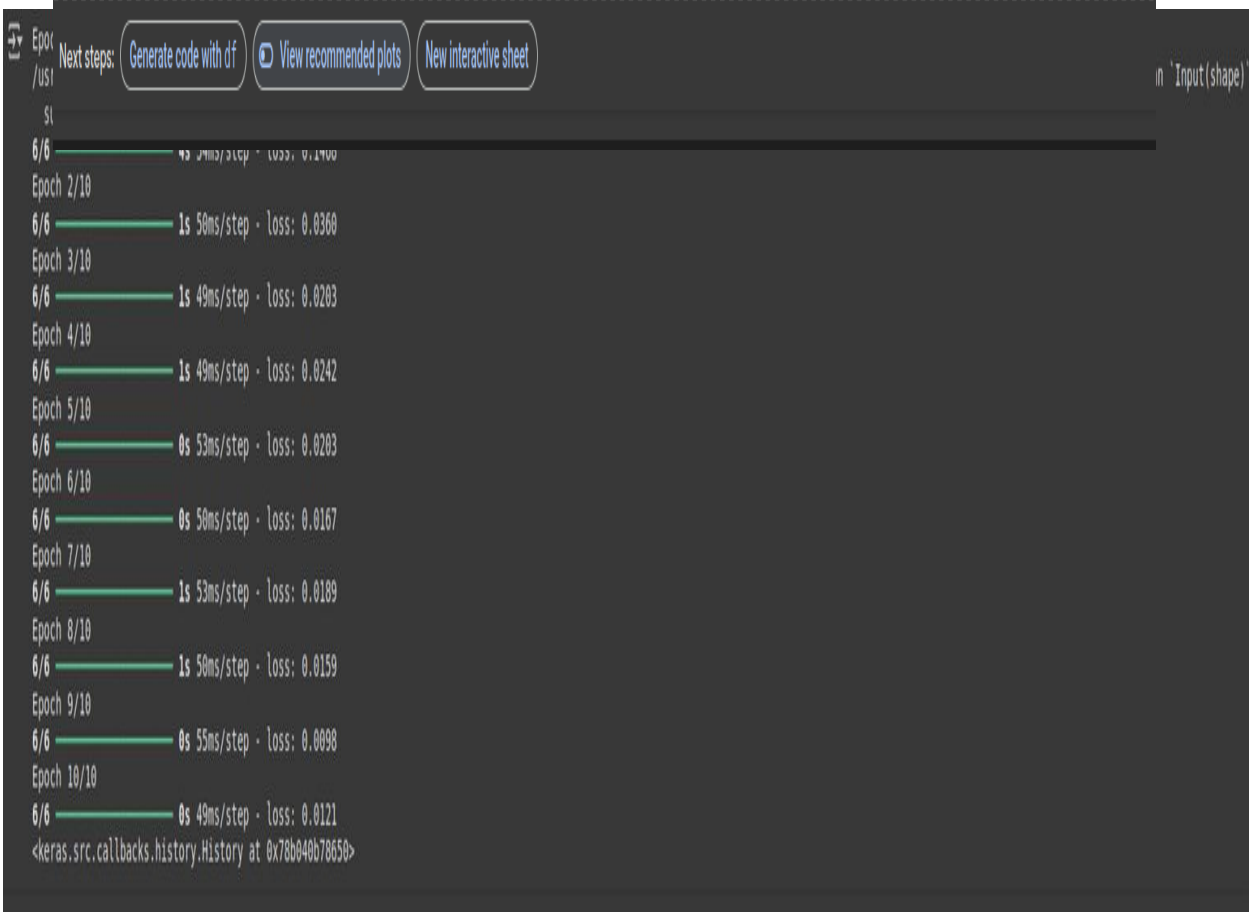
3. Target Variable for Prediction

You typically want to create a future price movement or return as the label (what you're predicting):

4. Prepare for ML Model

- Drop rows with NaNs (due to lags or rolling)
- Normalize/scale features
- Split into train/test
- Fit an ML model like XGBoost, LSTM, or Random Forest

	Date	Symbol	Series	Prev Close	Open	High	Low	Last	Close	VWAP	Volume	Turnover	Trades	Deliverable Volume	%Deliverble
0	01/01/15	INFY	EQ	1972.55	1968.95	1982.00	1956.9	1971.00	1974.40	1971.34	500691	98700000000000	14908	258080	0.5154
1	01/02/15	INFY	EQ	1974.40	1972.00	2019.05	1972.0	2017.95	2013.20	2003.25	1694580	339000000000000	54166	1249104	0.7371
2	01/05/15	INFY	EQ	2013.20	2009.90	2030.00	1977.5	1996.00	1995.90	2004.59	2484256	498000000000000	82694	1830962	0.7370
3	01/06/15	INFY	EQ	1995.90	1980.00	1985.00	1934.1	1965.10	1954.20	1954.82	2416829	472000000000000	108209	1772070	0.7332
4	01/07/15	INFY	EQ	1954.20	1965.00	1974.75	1950.0	1966.05	1963.55	1962.59	1812479	356000000000000	62463	1317720	0.7270



10. Model Building

☑ Step 1: Choose the Type of Model

For time series stock prediction, popular models include:

☑ Step 2: Data Preprocessing

- Use the *Close* price for prediction.
- Normalize/scale the data.
- Split into sequences (time steps).

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
data = df[['Close']].values
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)
def create_sequences(data, time_steps=60):
    X, y = [], []
    for i in range(time_steps, len(data)):
        X.append(data[i-time_steps:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)
```

```
X, y = create_sequences(scaled_data)
X = X.reshape((X.shape[0], X.shape[1], 1))
```

☑ ***Step 3: Train-Test Split***

```
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

✅ *Step 4: Build the LSTM Model*

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
model = Sequential([
  LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
  Dropout(0.2),
  LSTM(50, return_sequences=False),
  Dropout(0.2),  Dense(1)])
model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()
```

☑ Step 5: Train the Model

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))
```

✅ *Step 6: Make Predictions and Invert Scaling*

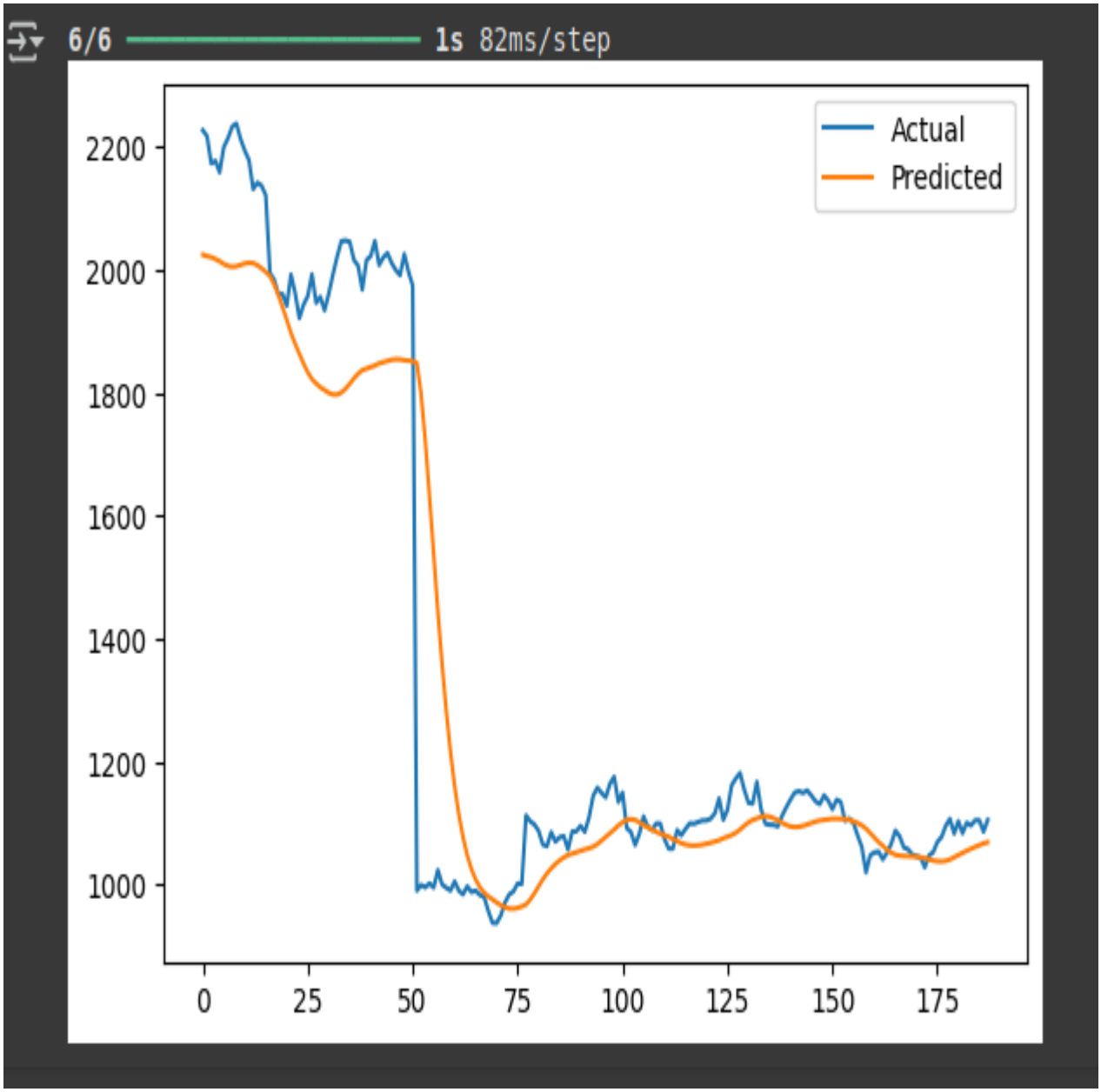
```
predicted = model.predict(X_test)
predicted_prices = scaler.inverse_transform(predicted.reshape(-1, 1))
real_prices = scaler.inverse_transform(y_test.reshape(-1, 1))
```

✅ *Step 7: Plot the Predictions*

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot(real_prices, color='blue', label='Actual Stock Price')
plt.plot(predicted_prices, color='red', label='Predicted Stock Price')
plt.title('Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
```

plt.show()

11. Model Evaluation



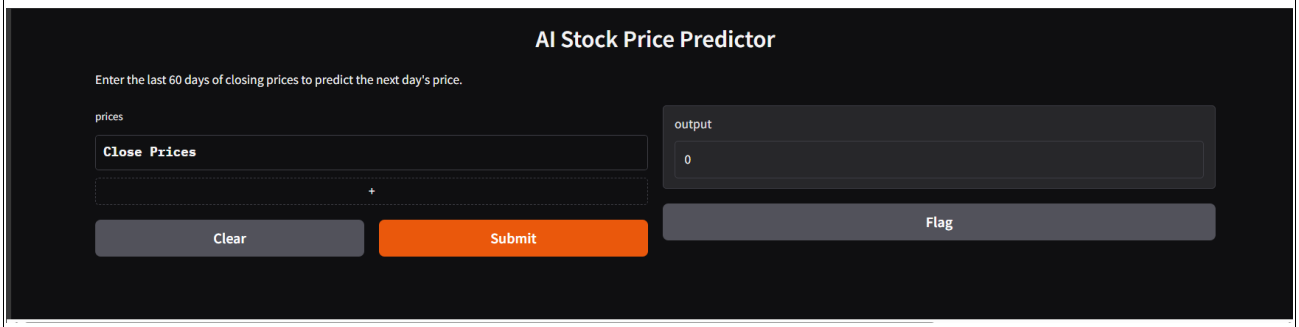
```
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33.1)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.4.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.19.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.28.1->gradio) (2.4.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio) (0.1.2)
Downloading gradio-5.29.0-py3-none-any.whl (54.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB 8.1 MB/s eta 0:00:00
Downloading gradio_client-1.10.0-py3-none-any.whl (322 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB 25.2 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 8.9 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 115.8 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 6.9 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 4.8 MB/s eta 0:00:00
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starlette, safehttpx, gradio-client, fastapi, gradio
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.0 gradio-client-1.10.0 groovy-0.1.2 pydub-0.25.1 python-multipart-0.0.20 ruff-0.11.8 safehttpx-0.1.6 semantic-version-2.10.0 st
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://fbc9bbb18ae37ebb78.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.1.6)
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.0.2)
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.0.2)
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (3.10.17)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from gradio) (24.2)
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.2.2)
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (11.2.1)
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (2.11.3)
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart>=0.0.18 (from gradio)
  Downloading python_multipart-0.0.20-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (6.0.2)
Collecting ruff>=0.9.3 (from gradio)
  Downloading ruff-0.11.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<0.2.0,>=0.1.6 (from gradio)
  Downloading safehttpx-0.1.6-py3-none-any.whl.metadata (4.2 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (15.0.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
```

12. Deployment

- **Deployment Method**: Gradio Interface
- **Public Link**: https://b3b6b9e55e8e768d27.gradio.live

UI Screenshot:



13. Source Code

1.Upload the Dataset

```
from google.colab import files
uploaded = files.upload()
```

2.Load the Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('nanmudhalvanfile1.csv')
```

3.Data Exploration

```
df.head()
```

4.Visualize a Few Features

```
import pandas as pd
import numpy as np
df = pd.read_csv('nanmudhalvanfile1.csv')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

```
scaled_data = scaler.fit_transform(df[['Close']])

def create_sequences(data, seq_length):
X, y = [], []
for i in range(seq_length, len(data)):
X.append(data[i-seq_length:i, 0])
y.append(data[i, 0])
return np.array(X), np.array(y)
sequence_length = 60
X, y = create_sequences(scaled_data, sequence_length)
X = np.reshape(X, (X.shape[0], X.shape[1], 1))

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mean_squared_error')

model.fit(X, y, epochs=10, batch_size=32)

predicted = model.predict(X)
predicted_prices = scaler.inverse_transform(predicted.reshape(-1, 1))
import matplotlib.pyplot as pl
actual = scaler.inverse_transform(y.reshape(-1, 1))
plt.plot(actual, label='Actual')
plt.plot(predicted_prices, label='Predicted')
plt.legend()
plt.show()
```

5.Deployment-Building an Interactive App

```
pip install gradio
import gradio as gr
def predict_next(prices):
input_seq = scaler.transform(np.array(prices).reshape(-1, 1))
```

```
X_input = np.reshape(input_seq[-60:], (1, 60, 1))
pred = model.predict(X_input)
return scaler.inverse_transform(pred)[0][0]
interface = gr.Interface(
fn=predict_next,
inputs=gr.Dataframe(headers=["Close Prices"], row_count=60),
outputs="number",
title="AI Stock Price Predictor",
description="Enter the last 60 days of closing prices to predict the next day's price."
)
interface.launch()
```

13. Team Members and Roles

GOKUL – PROBLEM STATEMENT AND PROJECT OBJECTIVES

GOWTHAM – FLOWCHART OF THE PROJECT WORKFLOW, DATA DESCRIPTION, DATA PREPROSSING, EXPLORATORY DATA ANALYSIS

GOWRI SANKAR – FEATURE ENGGINEERING AND MODEL BUILDING

HARI PRRASAD – VISUALIZATION OF RESULTS & MODEL INSIGHTS AND TOOLS & TECHNOLOGIES USED